

# The QXAS Editor

*Matthew Marcus*

September, 2010

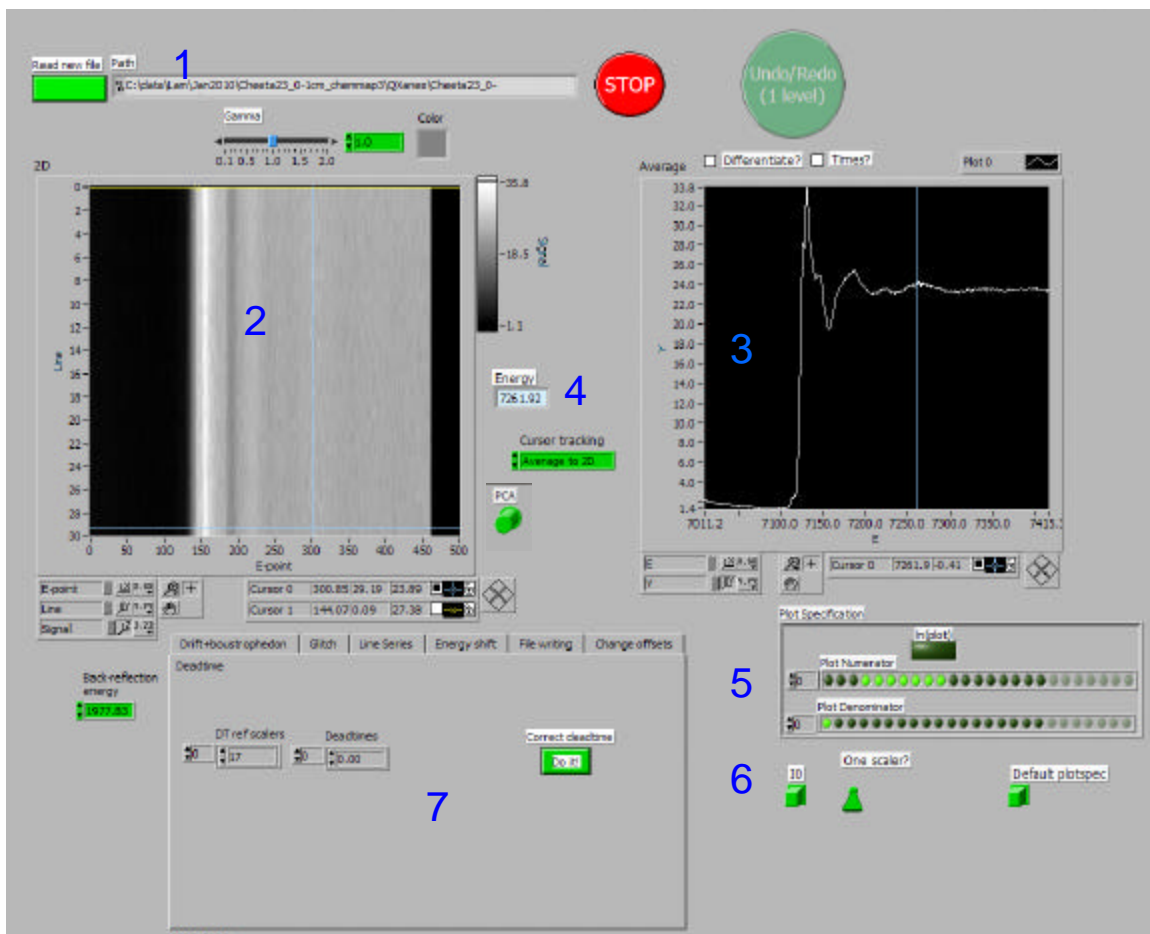
## **I. Introduction**

The QXAS Editor is the first-stage program for analyzing QuickXAS data. Many of the ideas are the same as in the EXAFS Editor, so if you're not familiar with that program, read its manual first. Each scan of such data is a set of monochromator scans, called lines, which can be plotted as a 2D intensity plot. Things one wants to do with such data include:

1. Display in various ways – as the intensity plot and as a normal XY graph, showing the average over some set of lines. One of course may want to display different combinations of scalers (plotspecs).
2. Corrections, including deadtime, energy drift from line to line, and any odd-even effect which may arise from boustrophedon scanning. This can also include shifting of energies and changes in offsets, as in EXAFS Editor.
3. Deglitching individual lines, which is often needed because of the problems the XIA hardware sometimes has with making transients in the data. Glitches that repeat at specific energies for all lines are best handled in EXAFS Editor.
4. Time-series plots. When studying kinetics or looking for radiation damage, one often wants to see what the data look like as a function of line number, averaged over some energy range. PCA provides a useful way to detect and understand time-series trends, so there's a subsystem for this similar to that in XE Reader.
5. File writing. One often wants to write a `.dat` file to be read into EXAFS Editor, or go directly to `.r`. Of course, one may want to write a corrected file as QXAS data, in `.qx` format. In addition, once you've made a time-series plot, you may want to save it in 2-column ASCII form, for which there's the new extension `.ser`.
6. Sum scalers. Sometimes you want to make a channel show the sum of some but not all of the detector channels. This is the same as the sum scalers into scaler function of EXAFS Editor.

## **II. Main screen displays and controls – data inspection**

The main screen, upon invocation and reading in a file looks like this:



The areas on the screen are:

1. File reading.
2. The 2D intensity plot. A gamma control is available as are two crosshair cursors. These cursors control the display of processed data. Note that the X-axis on this plot is not energy, but “E-point”, because the data are not in general uniformly tabulated in energy, but LabVIEW doesn’t have an obvious way to display 2D data with X and Y not uniform. Therefore, the data shown are relatively stretched in the XANES region, where the point spacing is finest in energy. This effect also occurs in the data-acquisition program.
3. The averaged-data plot. This shows the spectrum added up for lines between the yellow and blue cursors. As shown, it’s all lines, but by narrowing the gap between cursors, you can select out different sets, for instance, comparing early with late to see beam damage. The small checkboxes above the graph let you display the derivative or the dwell time.

4. Cursor tracking and energy indicator. The blue energy indicator shows what energy corresponds with the energy point on the intensity plot selected by the blue cursor (upper one, Cursor 0). As with the QuickXAS data acquisition program, you can have the cursor on the average plot track Cursor 0 on the intensity plot, the other way around, or have the cursors be independent.
5. Plotspec panel. Just as in EXAFS Editor, but the actual plotspec tends to be different from that of a normal-EXAFS file. What's shown is what the data-taking program was using, which in this case is the default. Change the settings here and the display and what gets written in an output 2-column file change accordingly.
6. Quick-setting controls for plotspec. The  $I_0$  button sets the plotspec so that  $I_0$  is displayed. The **Default plotspec** button restores to what was specified by the input file. The **One Scaler?** switch, if up, allows you to pick one of the scalers which will be the sole occupant of the **Plot Numerator** set. Thus, you can quickly rotate through scalers looking for Braggies or other glitches.
7. Operations tabs. These are where the buttons are that do things to the data. On entry to the program, this is always set to **Deadtime**. The next section of this manual describes what all the buttons and tabs do.

### III. Functions

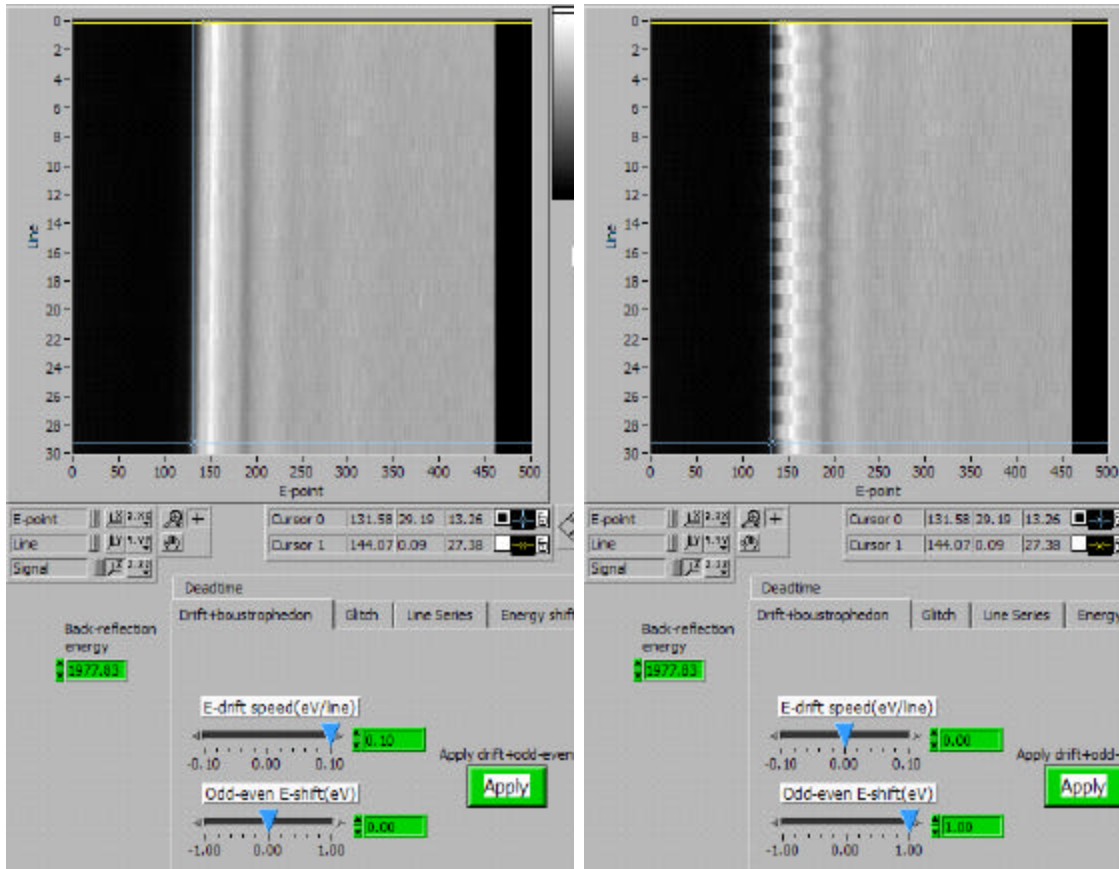
#### A. *Deadtime*

The first thing one usually wants to do is to deadtime-correct the data, which is done through the Deadtime tab shown in the above screenshot. Usually, the specifications for deadtime correction are correct as given and don't need to be adjusted. Just in case, here's what they mean: Consider scaler 3, which by default reads the counts in the fluorescence ROI of detector 0. The deadtime fraction is a function of the total counts coming into detector 0, which are represented by scaler 10. Thus, we say that the reference scaler for correcting scaler 3 is 10. Similarly, since scaler 10 is the one with the total counts, it is its own reference scaler. The deadtime in both cases is  $3.22\mu\text{s}$  for the normal detector configuration. Now, the  $I_0$  scaler is a bit special because it represents pulses coming from a V/F through a converter box into the 8<sup>th</sup> XIA channel. As such, it's not a Poissonian counter and doesn't obey the usual deadtime rules. Tests show that its response is pretty much flat up to a certain count rate (hence the rules about the divider switch), so it gets no deadtime.

This seems pretty complicated, but unless there has been a change of setup, it's simple to use. Simply press the DoIt! button. A flag in the file prevents you from double-deadtiming it. Even if fast detector mode is used, the scaler-map file should have been modified to change the deadtime numbers. If not, they can be modified on the Deadtime tab.

### B. *Drift+boustrophedon*

This tab is used to correct drifts of the mono from line to line (not yet observed) and the odd-even effect which comes from scanning forward and backward in boustrophedon mode. The effect of these functions on data which did not originally have such problems is shown here:

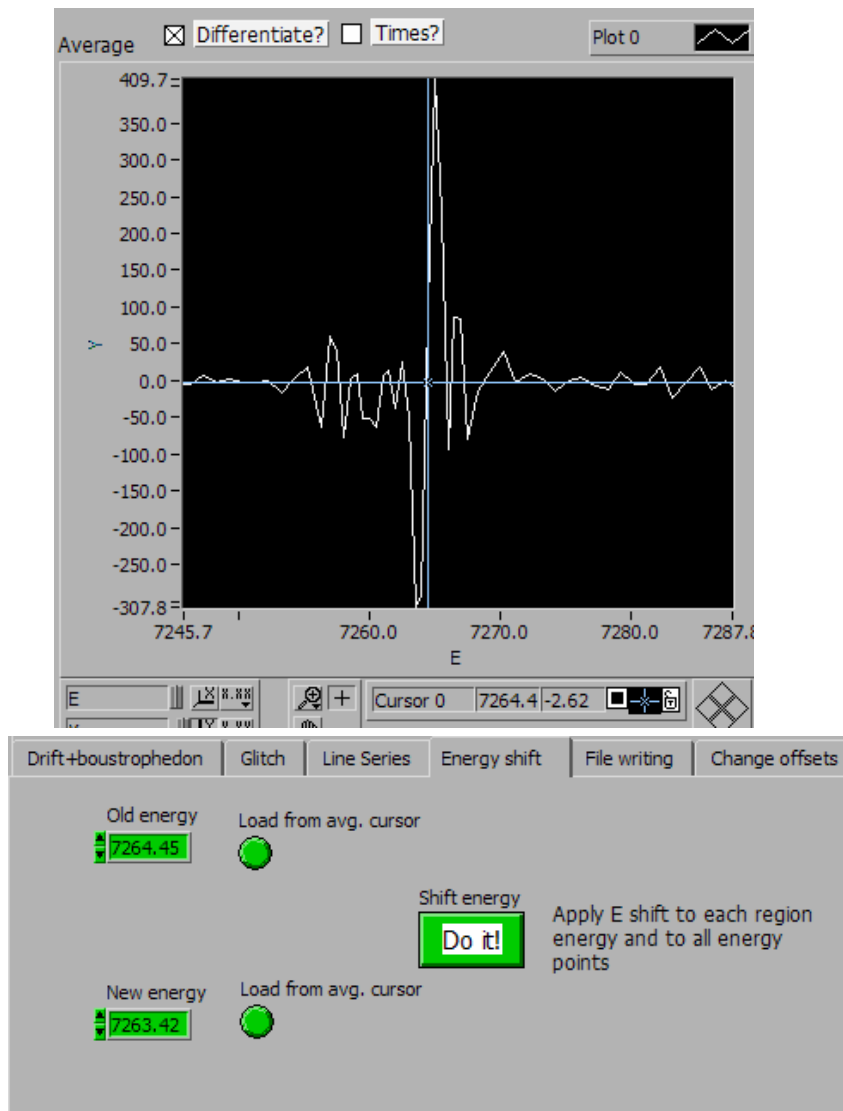


On the left, we see what happens if we assume a drift. Note the slant of the bright line with respect to the blue cursor. On the right is the effect of an odd-even shift. The above examples illustrate what happens when these corrections are applied to data which needs

no correction, but equally well show the “before” case for data which does need it. In any case, the average plot (not shown) reflects any correction applied. To make it “take” you have to hit the Apply and wait a few second for the correction to be applied to all scalers, not just the ratio or scaler shown in the plots. The sharpness of I0 glitches is often a good indication of the need for correction and how much to apply.

### C. *Energy shift*

A related operation is that of energy shifting, which is the same as what’s done in EXAFS Editor. This is used to adjust energy calibration, for instance putting a glitch or an edge right on a nominal value. The following picture shows an example:



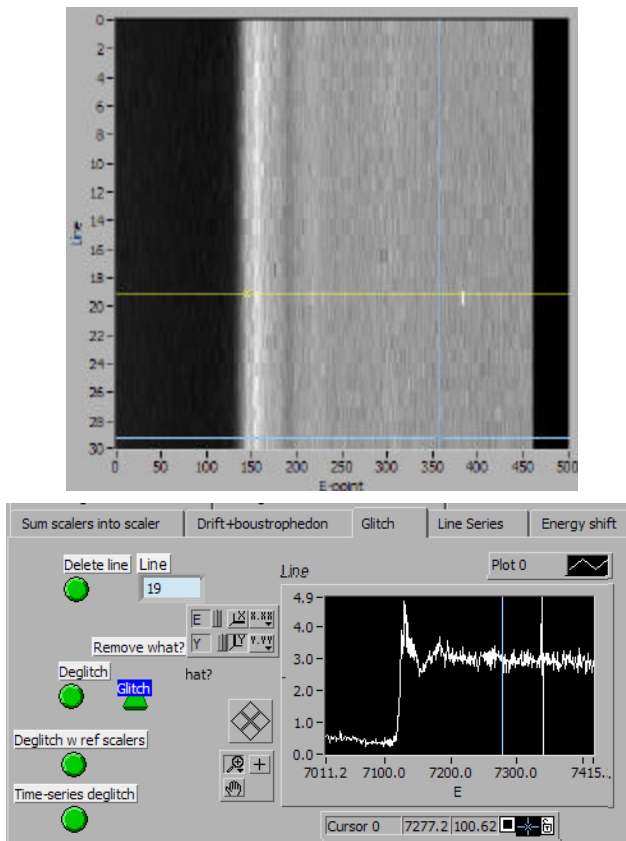
The top half of the picture shows the derivative of  $I_0$  and the cursor placed where it goes through 0, at the bottom of the big Fe-edge glitch. The bottom half shows the **Energy shift** tab after the **Load from avg. cursor** button was pressed, reading in where the glitch is. On hitting **Do it!**, the energies will be shifted so that the glitch appears at 7263.42eV.

#### D. *Change offsets*

The **Change offsets** tab is just like the one in EXAFS Editor, so needs no further discussion. Use it when you realize you forgot to redo offsets after changing a gain.

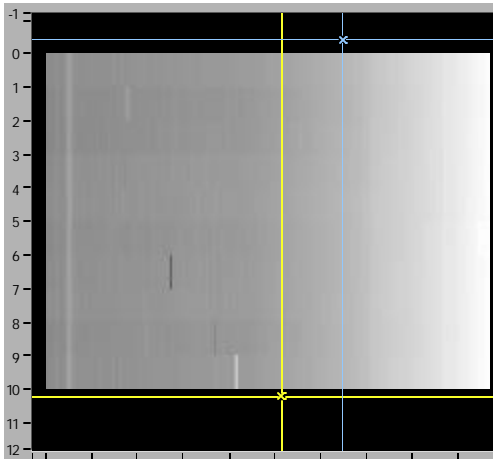
#### E. *Glitch*

Referring back to the first and second figures, a sharp-eyed reader may notice a bright tick in the intensity plot at line 19. This is a hardware-induced glitch analogous to the speckles one sees in XRF maps of uniform areas. A bit of exploring with the **plotspec** control shows that the problem is with scaler 8.

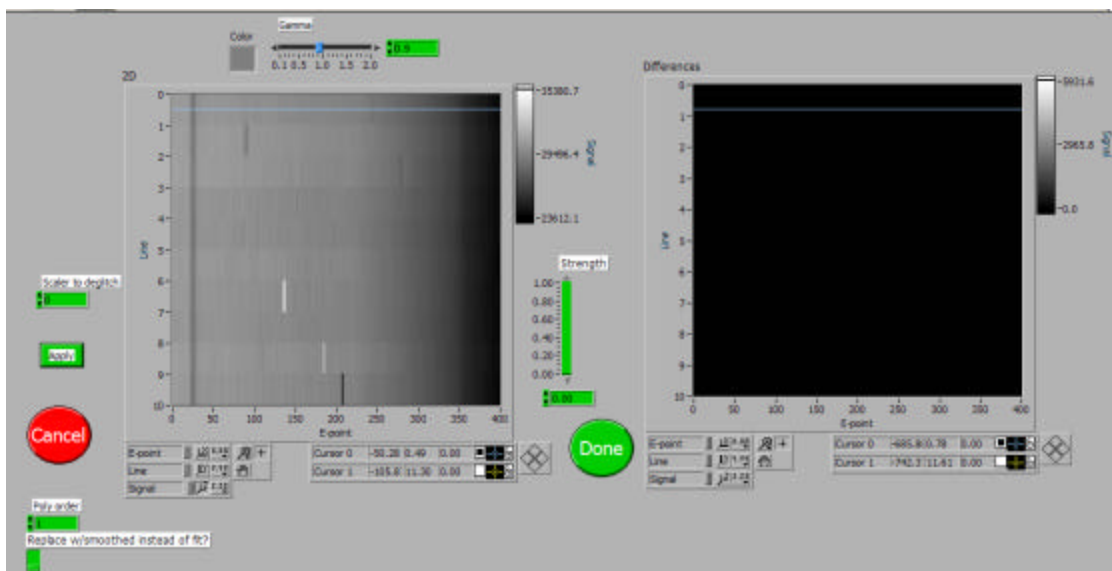


Here, the plotspec is set to  $8/I_0$  (scaler 8 /  $I_0$ ) and one of the cursors has been positioned so that it picks out the line on which that white feature occurs, line 19. The **Glitch** tab shows which line is being selected. It doesn't matter which cursor you move to select the line on which to operate; it looks at the last one you moved. The **Line** graph shows the selected data, making it obvious that there's a problem. The tab offers three ways to solve the problem. The most direct is simply to delete the offending line, which is what the topmost of the three green buttons does. The middle button leads you to a deglitch screen like the one in EXAFS Editor. The switch next to that button lets you do a jump correction instead of a deglitch. The display range of that screen is determined by the range of the **Line** graph, so zoom in on the relevant area in the **Line** graph before hitting **Deglitch** or **Deglitch w. ref scalers**. The latter button does the same thing as the corresponding function in EXAFS Editor, fitting ratios of scalers to polynomials and filling in the bad data for the numerator scaler (in this case 8) by assuming it to be given by the product of the denominator scaler sum and the fitted ratio. See the EXAFS Editor manual for more explanation of this choice.

Some channels, especially  $I_0$ , will show multiple glitches. Removing them one by one is tedious. It is therefore useful to have some way to detect and remove them automatically. That is the function of the **Time series deglitch** button. Here is an example of  $1/I_0$  with several glitches:



Pushing the **Time series deglitch** button takes you to this screen:



The left-hand panel shows the data for the scaler specified by the **Scaler to deglitch** control at the left. Since it's scaler 0 we want to correct, we leave that control alone. The **Strength** control, now set at 0, specifies how “aggressively” the program looks for presumed glitches. If it's at 0, nothing is done.

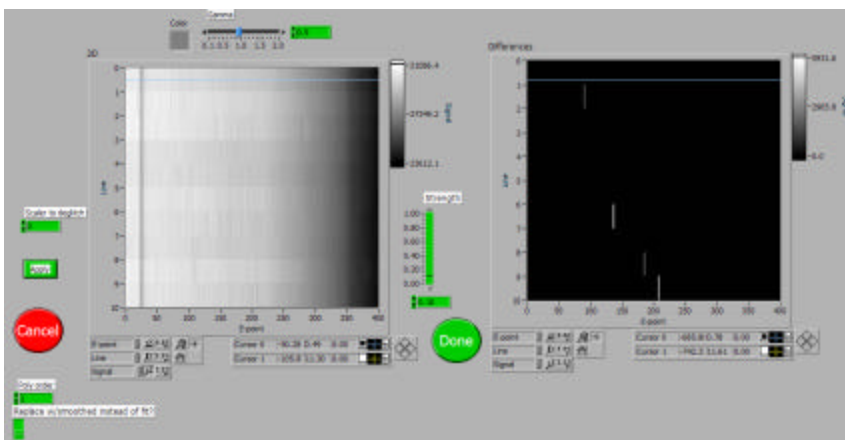
Before showing the actual deglitching, I'll explain what this screen actually does. The guiding assumption is that good (non-glitchy) data may be described by this equation:

$$I(\text{line}, \text{energy}) = c(\text{line}) * s(\text{energy}) \quad (1)$$

This formula implies that the XAS spectrum is the same for all lines, and what changes is an overall scale, an assumption which should certainly be true for  $I_0$  and at least approximately true for the other scalars. Any datum that deviates “too far” from this pattern is considered a glitch. The **Strength** control adjusts the sensitivity. Appendix 1 contains a description of the algorithm used. Any datum declared a glitch is replaced by either a smoothed version of the data for that energy, based on the data in other lines at that energy, or with a fitted form based on the entire dataset. For the most part, you can leave the controls under the **Cancel** button alone, though playing with them may improve results slightly.

Now let's look at what happens with a non-zero **Strength**:





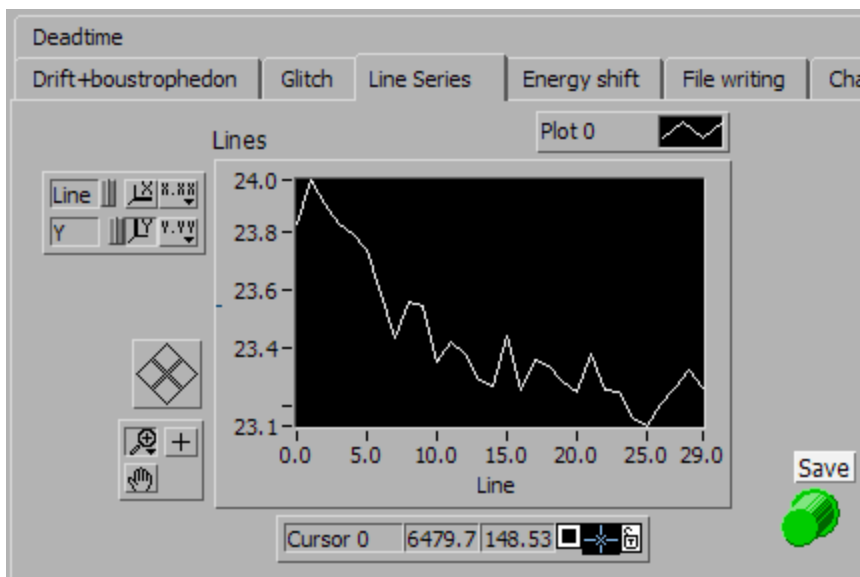
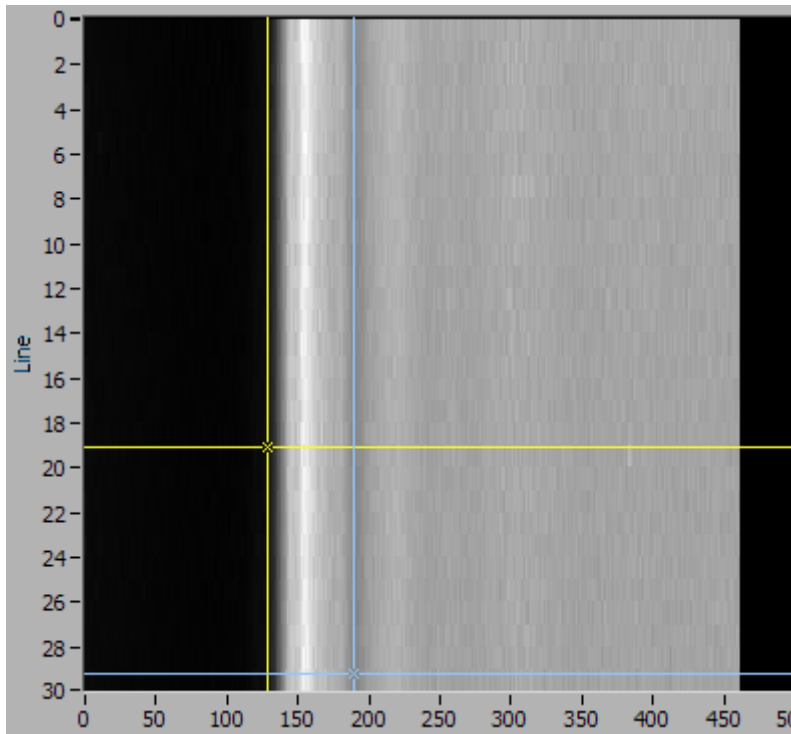
Here, I've moved the control just enough to make the obvious glitches disappear from the left-hand side. Note that the real mono glitch used for calibration, which appears as a dark line on the left side of the left plot, does not disappear. This is because the mono glitch appears on every line so is consistent with equation (1) above. The plot on the right-hand side is not blank and black anymore. It shows the absolute value of the difference between the deglitched and input data, so gives you an idea of what the program is doing. The **Gamma** control affects the color table for both plots.

If you like the result, hit the green **Done** button to get back to the main screen. If you're disgusted by the process, hit the red **Cancel** button to return without having done anything. If you want to check other scalers, hit **Apply**, which sets the deglitched scaler in memory. The **Strength** control resets to 0 and you can now choose other scalers to look at.

#### F. *Line series*

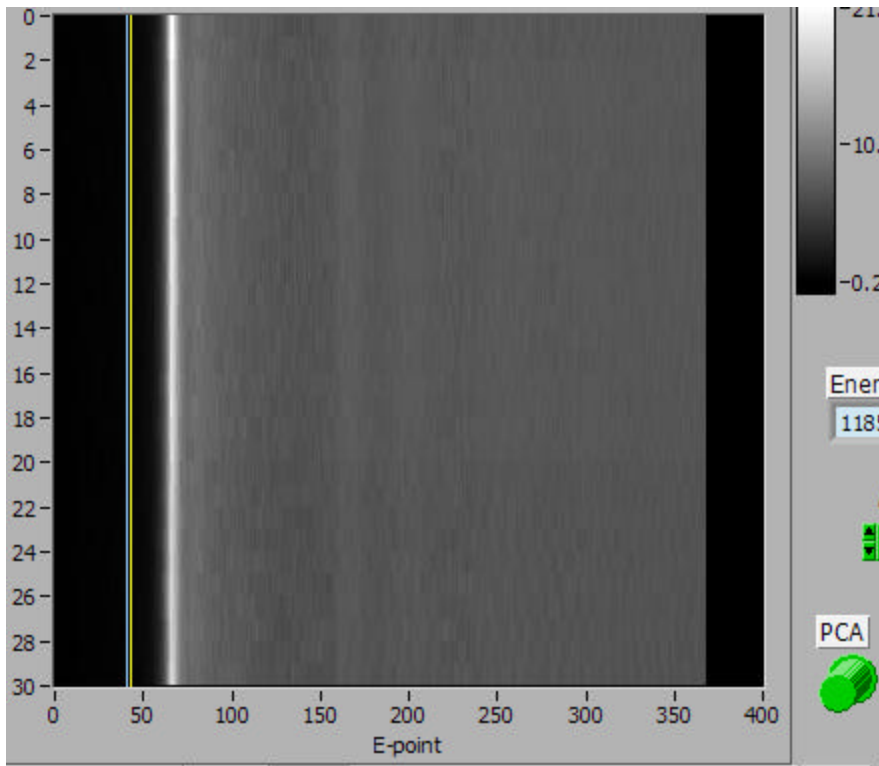
One of the reasons for doing QuickXAS in the first place is to study time dependence, either because it's of interest or because it signals a problem such as radiation damage or sample motion. It's therefore of interest to plot the data "the other way around", i.e. averaging over a range of energy and plotting versus line number. In the example below, the data are averaged between the pre-edge and the first dip, and the average plotted vs. line number. The energy range for averaging is given by the horizontal positions of the cursors. Since it's necessary to see where they are horizontally, the cursors are automatically set to the "+" style when this tab is displayed, then restored to whatever style they were before when one moves away from the Line Series tab. Contrary to how

the other tabs work, the vertical positions of the cursors are irrelevant here. The Save button saves the averaged data in a 2-column ASCII form with a `.ser` extension.

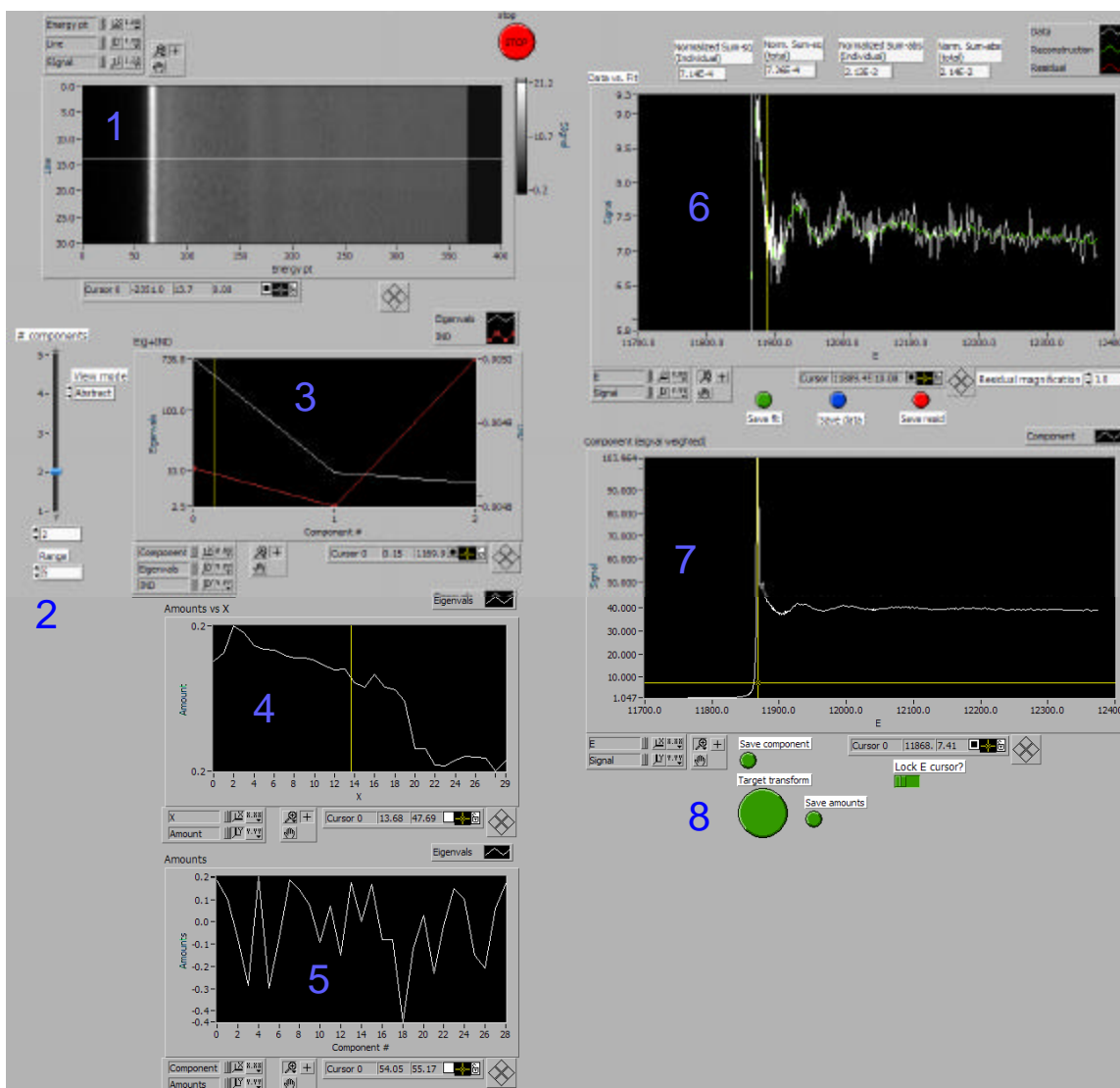


### G. PCA

This should, in principle, be part of the Time Series functions, but it has its own button to the right of the 2D graph, right under the **Cursor Tracking** control. In this, I'll assume knowledge of PCA and ITFA as used, for instance, in the **PCA w PRM** program. Let's take, as an example, data for which the 2D representation looks pretty good:



No obvious change here, but looking at the averages of lines 0-9, 10-19 and 19-29 reveal that there do seem to be trends with time. Pushing the PCA button reveals a new screen:



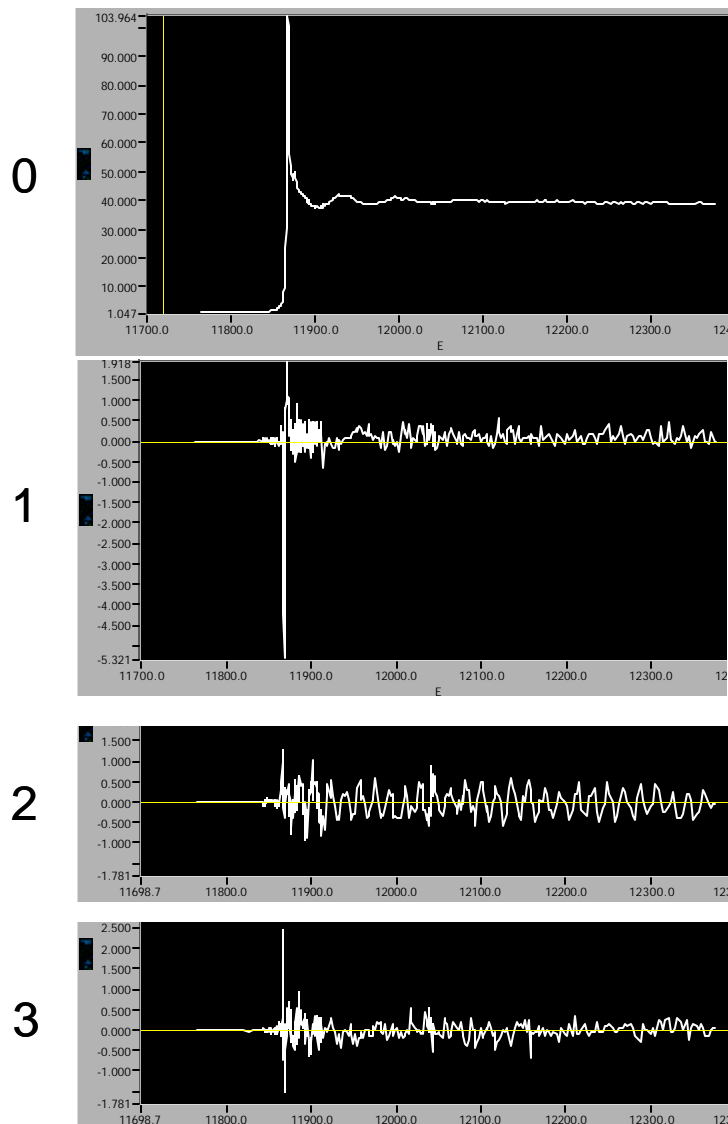
The relevant areas are:

1. The 2D data. Just like the left-hand plot in the main screen. There's one cursor, used to select which line to specify in some of the functions.
2. Number of components slider. Used to specify how many components ( $N_{\text{comp}}$ ) to use in the analysis. The **Range** control under it lets you specify the upper limit on the control.
3. This indicator shows the *IND* and eigenvalue (red and white) as a function of the number of components (actually,  $\#-1$ ). The cursor is used to select which component to examine. Component numbers start at 0 so that's why the abscissa goes from 0 to  $N_{\text{comp}}-1$ .
4. Amount vs. line number. This plots the loading of the specified component as a

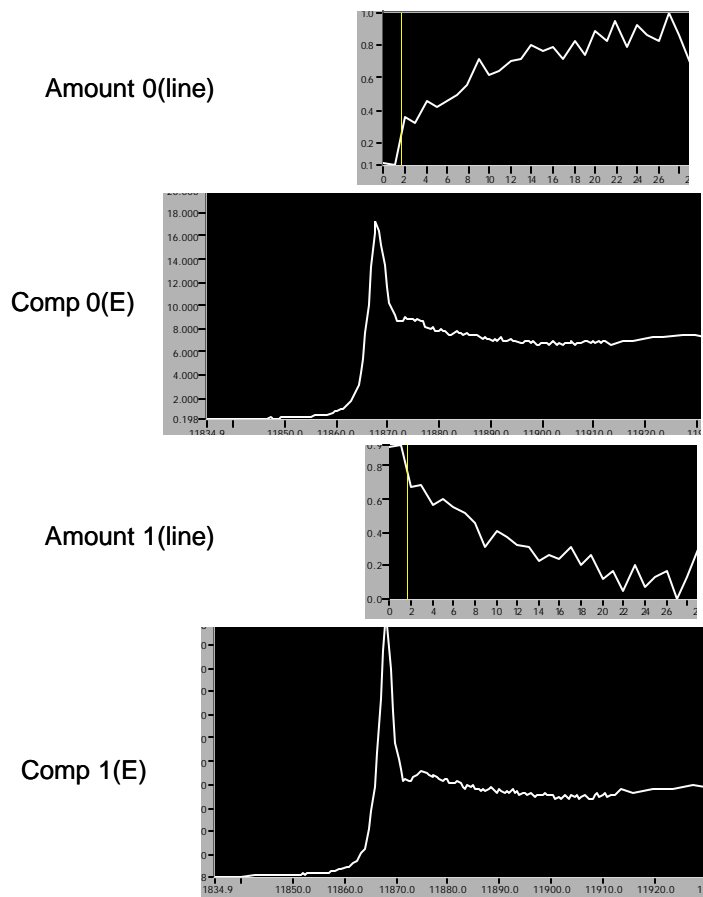
function of line number (called X here for no good reason). Thus, we see a rough trend of abstract component 0 (~ overall intensity) going down. Perhaps the sample is drifting a bit.

5. Amount vs. component number. This is the loading of the line specified by the cursor in area 1, as a function of component number. We're in Abstract view mode, so the loadings of all components fluctuate around 0 with magnitude comparable to 1.
6. The fit area. Shows the fit compared with the data. It fits pretty well with 2 components. As with the PCA program, this one lets you save the fit and residual, and plot the residual on a magnified Y scale.
7. Components screen. This is now showing Abstract component 0 (Abstract specified by the View Mode box in area 2, 0 chosen by the cursor in the *IND*/eigenvalue plot). Generally, Abstract 0 looks like an average of all the data.
8. Component save and Target test buttons. Just like in the regular PCA program. Plagiarism is no crime if you do it to yourself.

Now, let's assess whether there's a trend in the data. The *IND* value minimizes at 2 components. The Abstract components look like this, plotted one after the other:



These are the plots in area 7 (component) with the vertical scales all the same except for 0 and the Component cursor (area 3) set for 0,1,2,3. As stated above, component 0 looks like an average. Components 2 and 3 pretty much look like noise. Note the sinusoidal part of component 2. This is probably due to oscillatory sample motion which causes a modulation in the spectrum. Component 1, at least in the white-line region, at least looks like it might be real signal. The fits get better when you include it and look carefully, but not hugely so. However, ITFA mode tells a story. Switching the View mode to ITFA and setting the # Components slider to 2 yields a pair of ITFA components and amounts for each:



$$\text{Signal}(\text{line}, E) = \text{Amount0}(\text{line}) * \text{Comp0}(E) + \text{Amount1}(\text{line}) * \text{Comp1}(E)$$

I've broken out the Amount and Component graphs separately. The upper pair is how it looks with the cursor on the Eig+IND panel set for the 0 component, and the upper pair is for component 1. The Comp graphs have been zoomed in to make the white lines visible. You can see that the two components have somewhat different white-line peak heights and that the dip just after the white line is less pronounced in Comp0 than Comp1. This could still be noise of some sort except for the fact that there's a consistent trend per-line in which Comp0 comes up and Comp1 drops, in what looks qualitatively like a first-order kinetic pattern. It's hard to see what artifact would cause such an appearance except perhaps if the area at which we were looking were inhomogeneous and the sample drifted, quickly at first, then slowing down, which is possible. In any case, we see how PCA can be used to pull out a non-obvious trend. All the relevant quantities such as amounts can be saved by pushing various buttons. The red STOP button at the top of the screen puts you back in the main screen.



#### *H. File writing*

This tab shows buttons labeled QXAS, EXAFS (.dat) and 2-column (.r). Pushing one of these buttons saves a file of the indicated type of data, with extension .qx, .dat or .r, respectively. The QXAS file reflects all data-changing operations which have been performed so far.

## Appendix I – the time-series deglitch algorithm.

The first step is to fit the data  $I(line, energy)$  to the form  $c(line)s(energy)$ , on a line-by-line basis. Define

$$\bar{I}(energy) = \langle I(line, energy) \rangle_{line} \quad (2)$$

where the angle brackets mean an average over a variable. Now, the form (1) is ambiguous in that you can multiply one factor by anything and divide the other by the same amount. I will define  $s(energy) = \bar{I}(energy)$ , which requires in a least-square sense that

$$c(line) = \frac{\langle I(line, energy) \bar{I}(energy) \rangle}{\langle \bar{I}(energy)^2 \rangle} . \quad (3)$$

Next, define the normalized signal  $N(line, energy)$  as

$$N(line, energy) = I(line, energy) / c(line) \quad (4)$$

which, in the absence of noise or glitches, would be the same for all lines and depend only on energy. Now, for each energy, we want to examine this normalized data and look for what isn't the same for all lines. To do this, we fit  $N$  to a polynomial in line number, yielding  $N_{fit}(line, energy)$ , then take the residual. The criterion for a data point to be considered a glitch is then

$$glitch = (N - N_{fit}) > S / S \quad (5)$$

where  $S$  is the “strength” and  $S$  is the standard deviation of the residual  $N - N_{fit}$  at the particular energy being considered. This criterion is evaluated for each energy.

Now, suppose a glitch is found at  $(line_g, energy_g)$ . There are obvious ways to “mend” the data at that point. One way is to replace it using the polynomial fit to  $N$ , so that

$$I(line_g, energy_g) \leftarrow c(line_g) N_{fit}(line_g, energy_g) \quad (6)$$

and the other is to replace it with the value of the fit from (1):

$$I(line_g, energy_g) \leftarrow c(line_g) \bar{I}(energy_g) . \quad (7)$$

It is not clear to me which method is best under which circumstances, so both are provided.